REMARKS

Claims 1-22 are in this case. Claims 1-22 have been rejected. Claim 1 has now been amended.

Rejections Under 35 U.S.C. 102(e) – Edwards

The Examiner has rejected claims 1-22 under 35 U.S.C. 102(e) over US Patent No. 6,625,797 to Edwards et al. ("Edwards"). The rejections of the Examiner are respectfully traversed in light of the above amendments to claim 1.

It is noted that the Examiner commented in the Response to Arguments section in the Office Action that no patentable weight was given to the feature "structural constraints", since this feature appeared only in the preamble. The feature of structural constraints, which is not shown in the prior art, has now been transferred to the body of the claim, along with other features that help to define a verification language. It is now believed that the feature attracts patentable weight and allows for the claim to be distinguished over the prior art which, whilst defining a first language, does not define a first language having such structural constraints. As defined in the claim, it is the structural constraints which determine the relationships between objects in the object hierarchy. The object hierarchy is the framework in which the dynamic behavior of the code takes place and, as will be explained below, it is such relationships which make it difficult for the prior art to be able to map the dynamic behavior of the code. Since the prior art neither knows nor hints at the existence of such structural constraints it is believed that the mapping from an originating language having such constraints is novel and inventive.

Firstly we provide a brief summary of Edwards. As discussed in the previous response, Edwards teaches a method for efficient compiling software for various hardware

implementations. This method is stated to allow the software programmer to work at a high level of abstraction, while the machine-dependent (hardware-dependent) details of the program are handled according to the taught method, without requiring direct intervention by the programmer. The programmer constructs the program according to a high level of abstraction, while a semantic model handles the hardware-dependent aspects of the program. The semantics are interpreted through a tool which creates a control and flow data graph as part of the compilation of the software. The method of Edwards is specifically taught as being useful for such computer programming languages as C and C++, which are high level software imperative programming languages but crucially do not include structural constraints. On the contrary, it makes no sense for a general purpose imperative programming language to include structural constraints because a general purpose programming language lacks any mechanism for dealing with or resolving such constraints. It is only when it comes to design verification (or some other highly specialized areas) that it makes sense to include structural constraints, as a modeling aid that is not reliant on imperative programming.

The present invention is of a system and method for synthesizing a first language for compilation into a target language, for the purpose of design verification. The present method, according to amended claim 1, defines the first language as one having *structural constraints for the purpose explained above*, dynamic behavior, an object hierarchy and relationships between objects defined within the object hierarchy. Such a definition is believed to exclude the high level programming languages intended in Edwards.

The method enables the underlying control structure of the verification language to be determined, and then used to map the dynamic behavior of the verification language onto the target language as part of a static framework. Verification languages have a number of characteristics which make such a process difficult. For example, a verification language

features the structural constraints, dynamic behavior, and a hierarchy of objects, as stated in claim 1 and expressed above. Structural constraints determine the relationships between the objects in the hierarchy of objects, and therefore need to be properly resolved and represented in the target language in order to map the dynamic behavior of the verification language for conversion to the target language.

Determining the static framework is referred to as "elaboration" in the description, is defined at the top of page 8 and is explained in some detail, particularly between pages 8 and 13. The discussion includes that of a resolution mechanism for structural constraints on page 11. The elaboration algorithm is shown in Fig. 2.

This process is not simple, and the present invention deals with the above issue by mapping the dynamic behavior as part of a static framework, this now being clearly stated in claim 1, and the result is to retain the functionality or capabilities of the first (verification) language for the target language.

The first stage in preferred embodiments of the present invention is the elaboration of an instance tree. This is a key process as described with regard to Figure 1 and the accompanying textual description. This elaboration is done based on the structural constraints now patentably defined in claim 1. The elaboration of the instance tree enables the method of the present invention to handle a plurality of objects *where structural constraints define relationships therebetween*, each having some associated storage and some function.

The elaboration of the instance tree is significant, since each object in the instance tree may have its own storage and processes, which will have to be translated to the target language; not every constraint may be resolvable at every level, as described with regard to Figure 1. The method also optionally includes construction of an elaboration graph (as a way of performing the elaboration process) and a data/control graph. The construction of both

graphs is preferred because of the nature of the verification language as containing a hierarchy of objects, with structural constraints determining relationships between the objects.

Thus, Edwards does not teach a method for compiling languages which feature "structural constraints". Rather he mentions only a control and data flow graph. The languages described in Edwards do not have such structural constraints, which are now defined in the body of claim 1. Edwards gives examples such as C++ and Java, which are high-level software languages. High-level software languages do not feature constraints and do not feature multiple objects which have relationships determined by constraints, for the reasons given above. In other words high level software programming languages are not equivalent to verification languages, which have a number of defined characteristics as follows: structural constraints, dynamic behavior, and a hierarchy of objects. In order for the method of the present invention to be operative, it must be able to handle languages with these characteristics, which are now recited in the body of claim 1 in such a way as to give them patentable weight.

By contrast, the method of Edwards does not teach or hint at a way of mapping for a language that allows multiple objects in a hierarchy whose relationships are defined by *structural constraints*. Therefore, the method of Edwards would not be effective for languages as defined in claim 1, which do contain *multiple objects whose relationships are defined by structural constraints*. The method of Edwards does not and cannot handle such issues as determining which instances exist, interaction patterns between independent objects, race detection, scheduling etc, because of the inability of Edwards to handle multiple objects in this way.

By contrast, the method of the present invention is able to handle the complexity arising from *a multiplicity of objects whose relationships are defined by structural constraints,* for example by determining the elaborated roles (storage), deriving the access patterns (creation of the conflict graphs), conflict resolution and scheduling. The method of Edwards cannot handle these issues and would therefore be inoperative for the languages as defined in amended claim 1 of the present application.

The present invention also optionally features an elaboration graph as well as a data/control flow graph, while Edwards only discloses the latter and indeed would only be operative with the latter type of graph.

In addition, claim 1 recites the creation of a static framework of resources to support the dynamic behavior from the first language as it is mapped onto the target language. The Examiner states in his response to arguments that a mere statement of non-equivalence is not persuasive. Edwards teaches a hardware-implementation independent flowgraph and the use of the flowgraph in assigning hardware resources. By contrast the claim defines a creation of a static framework of the resources in the first language and the use of that framework in the mapping of the dynamic behavior onto the second language. As explained in the passage bridging pages 6 and 7 of the specification, the static framework which is defined for the first language is expanded and used as the basis for the mapping into the second language, *in contrast to* the more conventional alternative of executing the first language and simply mapping the resulting functionality onto the second language. It is respectfully submitted that, contrary to the Examiner's assertion that a static framwork of resources is functionally equivalent to the hardware-implementation independent flowgraph of Edwards, a *static* framework of the resources of the first language *is* patentably distinct from the flowgraph of Edwards, since a *flow*graph with emphasis on the word "flow" is *dynamic*. That is to say

Edwards effectively carries out exactly what the present disclosure on page 7 excludes, which is to capture the dynamic behavior of the code *directly*. In Edwards, the bytecodes define the actual hardware circuits and these are derived from the flowgraph, which is the dynamic behavior as a series of nodes. Crucially the creation of the flowgraph is the first stage in Edwards. The step of the creation of the static framework of resources is entirely missing in Edwards, and for a very good reason. Since structural constraints do not define the relationships between objects, a static framework *cannot be derived* without resorting to the interpretation of the dynamic behavior. It is stressed again that the present invention is concerned with design verification where structural constraints are a crucial aspect of modeling. The static framework is therefore crucial in this context and it must be a part of the mapping process, whereas in the case of Edwards the idea is merely to translate the algorithms as accurately as possible for implementation. No verification is intended and therefore static frameworks and structural constraints are not of relevance.

In this connection the Examiner is referred to the example set forth on pages 31 to 32 of the instant application, for which the results are presented in Fig. 12. Fig. 11 includes a representation of structural constraints.

It is also pointed out that the elaboration algorithm includes a role for the structural constraints, viz the set C of yet unresolved constraints). Edwards does not do this at all.

Therefore, the method of the present invention, as recited in the claims of the present application, is clearly different from the method of Edwards, which would be inoperative with languages as defined in claim 1. Thus, claims 1-22 are clearly novel and non-obvious over Edwards.

From the above remarks and amendments, Applicant feels that claims 1-22 are now in condition for allowance. Prompt Notice of Allowance is respectfully and earnestly solicited.

Respectfully submitted,

D'vorah Graeser
Agent for Applicant
Registration No. 40,000

Date: November 4, 2004